CS 494 Object-Oriented Analysis & Design

Using PARTS to Illustrate Requirements Concepts

Reminder: Requirements

- Defining what the system should do
  - What the clients needs (as opposed to wants)
  - Not how the solution should be designed or implemented
- We recognize three iterative activities :
  - Elicitation: capturing information from sources
  - Documentation: "putting it on paper"
  - Validation: confirming it meets users' needs
- *Analysis* (or definition) versus *Specification*
  - Customer-oriented requirements
  - Develop-oriented requirements

BTW… Specification Documents

- Steven McConnell (IEEE *Software,* Oct. 2000) says any of the following are called "requirements document":
  - Half-page summary of software *product vision*
  - Two-page key *features list*
  - 50-page list of details about end-user requirements (he calls this a *function-requirements document*)
  - 250-page exhaustive list of details about screens and GUI, input and input conditions, all system states and state changes, all persistent data, etc.
- This 4th item is what we usually mean by a Software Requirements Specification (SRS) document

Examples based on PARTS

- Proposed software system:
  Project Artifact Report Tracking System (PARTS)
- PARTS' concept is very similar to commercial defect-tracking tools
- See "Vision Statement" for product concept
- Briefly, PARTS...
  - Helps a development team collect info on work-products (e.g. requirements document, design diagrams, code files, etc.)
  - Includes status and problem reports for an artifact
  - Knows about projects, team-members

PARTS is:

- A CASE tool for storing and tracking problem reports
  - Each report contains a problem description and a status
  - Each problem can be assigned to someone
  - Problem reports are made on one of the "artifacts" of a project
  - Employees are assigned to a project
  - A manager may add new artifacts and assign problem reports to team members

PARTS Example: Needs vs. Wants

- Customer says: "I want a client and a server developed in Java."
- Real need:
  - Centralized data store
  - Remote access by team members
- Other possible solutions:
  - Web pages and cgi-bin scripts
  - Commercial database products that support client access
  - Buy a commercial product!

PARTS Example: Domain, Constraints

- **What's the** domain **for PARTS?**
  Team-based Software Development
- **Domain vocabulary:**
  - **Work-product, artifact (what's the difference?)**
  - **Problem reports, project, team members**
- Domain dictionary **or Glossary: Frequently an output of the requirements activity**
- ***Possible* examples of** Constraints:
  - **System must use Oracle DBMS.**
  - **System must create MS Word reports.**
  - **System must be written in Java.**

PARTS Example: System Boundary

- **Different types of Users of the system?**
  - **Manager: Can create projects, assign a problem to a team-member**
  - **"Ordinary" team-member: Can access info, but not create projects, assign problems, etc.**
- **Hardware components?**
  - **Interaction with printer subsystem of the OS**
- **Other system entities:**
  - **Oracle DBMS, MS Word**
  - **Client-server communications using sockets**

Objects, Functions and States

- **Before continuing, consider another way of thinking about requirements…**
- **Alan Davis says: All requirements**
  - **Define an object, function or state;**
  - **Limit or control actions associated with an object, function or state;**
  - **Define relationships between objects, functions and states.**
- **The challenges:**
  - **Identifying these.**
  - **Representing and documenting them effectively.**
  - **Making use of this information later in development.**

Objects

- **Note: Davis' discussion attempts to include both OO and non-OO views of requirements**
- **What's an Object?**
  - **A real-world entity**
  - **Important to the discussion of requirements**
  - **Has a crisply-defined boundary**
- **Object's have attributes, functions, states, and relationships**
- **(Sometimes) Objects are groups into classes**

Functions

- **A task, service, process, activity, mathematical function, etc. that…**
  - **Is performed in the real world**
  - **Is to be performed by the system to solve the real-world problem**
- **Requirements about functions may**
  - **define, limit, specify relationships, etc.**
- **Functions may be group hierarchically**
  - **Abstract to specific (detailed)**
  - **Important: This is not design!**
    - **Organizing functions only for understanding requirements.**

States

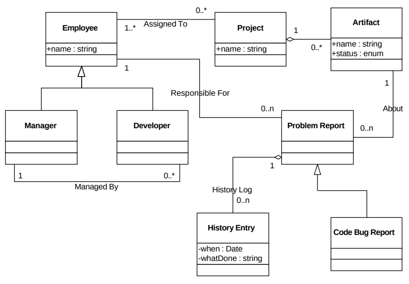
- **A condition of some** *thing***...**
  that captures some history of that thing...
  and is used by the thing to determine behavior.
- **What's a "thing"?**
  - **The system**
  - **An object**
  - **A function**

PARTS Example: Objects

- Objects the system must “understand”
  - Project, Artifacts
  - Team-member (with user-id and password?)
  - Problem report

2/6/01 D-13

Class Diagram for Prob. Rep. Tool



2/6/01 D-14

PARTS Example: Functions

- At what level?
  - (High-level) Enter a report for a given artifact.
  - (Lower-level) Prompt user to confirm request to delete a problem request
- (Note: *use cases* focus at high levels)
- Function classification and/or hierarchy:
  - Manager operations vs. ordinary operations
  - Operations related to queries and reports

2/6/01 D-15

PARTS Example: States

- System-level states:
  - Operations or interface available if a manager logs into PARTS
- Object states:
  - A problem-report can be *unassigned*, *open* or *closed* (i.e. resolved)
- Function states:
  - Possibly an command-history list for Undo and Redo
    - Perhaps some actions cannot be undone?
  - Non-PARTS example:
    a database transaction may be complete, in progress, aborted, etc.

2/6/01 D-16

PARTS Use Case Model: Actors

- Manager
  - A person assigned to a project with permission to do more things than an ordinary team-member
- Super User
  - Has the ability to create projects and users
- Member
  - An “ordinary” member of a development team
- Non-member
  - A user not assigned to a team who has been given read-access to a project by its manager

2/6/01 D-17

PARTS Use Case Model: Use Cases

- Let’s organize these by categories:
  - Project management related use cases
  - Problem Report related use cases
  - “Support” use cases
- In the next slides, we’ll list use case titles and the actors who participate in them
  - Even just doing this raises some good questions about imprecise requirements!

2/6/01 D-18

PARTS Use Cases: Management

- Create User (Actors: SU, Mgr)
- Update User Info (SU, Mgr, Member)
  - Let’s say “update” includes “delete”
  - Members can only update certain info about themselves
- Create Project (SU)
- Update Project (SU, Mgr)
- Add Member to Project (Mgr, SU??)
  - Hmm, do the requirements say the SU can do this?
- Create Project Artifact (Mgr, SU??)
- Update Project Artifact (Mgr, SU??)

PARTS Use Cases: PR-related

- Create PR for Artifact (Member, SU?)
- View PR (Member, Non-Member)
- Change PR Status (Member, Mgr, SU?)
- Update PR History (Member)
  - System does this too! Do we model this as part of the use case? Not obvious how!
- Assign PR to Member (Mgr)
- Delete PR (Mgr)
- Search for PRs (Member, Non-member)

PARTS Use Cases: “Support”

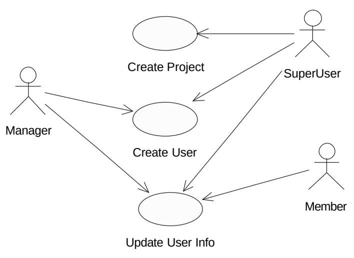
- Display Projects
- Display Project Artifacts
- Display Artifact PRs
- Log Into PARTS

- Comments:
  - All of these are “used” by other use-cases (perhaps)
  - Or, are these just parts of the user-interface
  - Need mechanism to look at and select a “thing”

PARTS Use Case Details

- On the Web site:
  - More detailed examples of use cases based on use case templates showing scenarios, etc.

PARTS UML Use Case Diagram


Create Project
SuperUser
Manager
Create User
Member
Update User Info